

9장. 기초 보강

(주교재 9장과는 다른 내용입니다)

래퍼클래스 소개, 컬렉션 설명, ArrayList 소개

동아대학교 컴퓨터공학과

우선 Wrapper Class에 대해
간단히 알아봅시다!!

Wrapper 클래스

- 자바의 기본 타입을 클래스화한 8개 클래스

기본 데이터 타입	byte	short	int	long	char	float	double	boolean
Wrapper 클래스 타입	Byte	Short	Integer	Long	Character	Float	Double	Boolean

- 용도

- 기본 타입의 값을 사용할 수 없고 객체만 사용하는 **컬렉션** 등에 기본 타입의 값을 Wrapper 클래스 객체로 만들어 사용

세션에서 `getAttribute`할 때도 반환 타입이 `Object`라서,
`String`, `Integer` 등의 클래스 객체로만 변환 가능
`int count = (int)session.getAttribute("count");` (X)

Wrapper 객체 생성자 호출

- 기본 타입의 값을 인자로

원래는

```
Integer i = new Integer(10); Character c = new Character('c');  
이런 식으로 생성자 호출
```

지금은 자동 박싱/언박싱으로 인해 그냥 `Integer i = 10;` 넣어도 됨

- 문자열 써도 됨

```
Boolean b = new Boolean("false");  
Integer l = new Integer("10");  
Double d = new Double("3.14");
```

주요 메소드

– 가장 많이 사용하는 Integer 클래스의 주요 메소드

메소드	설명
<code>static int bitCount(int i)</code>	인자 <code>i</code> 의 이진수 표현에서 1의 개수를 리턴
<code>float floatValue()</code>	<code>float</code> 타입으로 변환된 값 리턴
<code>int intValue()</code>	<code>int</code> 타입으로 변환된 값 리턴
<code>long longValue()</code>	<code>long</code> 타입으로 변환된 값 리턴
<code>short shortValue()</code> 가장 많이 사용됨	<code>short</code> 타입으로 변환된 값 리턴
<code>static int <u>parseInt(String s)</u></code>	스트링 <code>s</code> 를 10진 정수로 변환된 값 리턴
<code>static int parseInt(String s, int radix)</code>	스트링 <code>s</code> 를 지정된 진법의 정수로 변환된 값 리턴
<code>static String toBinaryString(int i)</code>	인자 <code>i</code> 를 이진수 표현으로 변환된 스트링 리턴
<code>static String toHexString(int i)</code>	인자 <code>i</code> 를 16진수 표현으로 변환된 스트링 리턴
<code>static String toOctalString(int i)</code>	인자 <code>i</code> 를 8진수 표현으로 변환된 스트링 리턴
<code>static String toString(int i)</code>	인자 <code>i</code> 를 스트링으로 변환하여 리턴

→ `i + ""` 과 동일 효과

Wrapper 활용

- 문자열을 기본 데이터 타입으로 변환

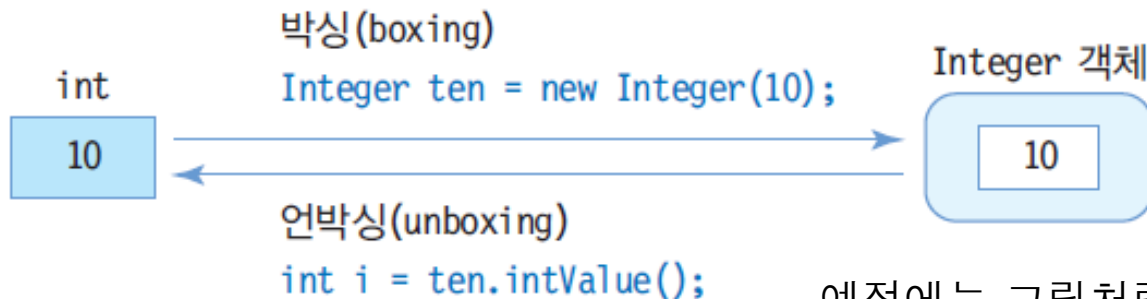
```
int i = Integer.parseInt("123"); // i = 123  
boolean b = Boolean.parseBoolean("true"); // b = true  
float f = Float.parseFloat("3.141592"); // f = 3.141592
```

- 기본 데이터 타입을 문자열로 변환

```
String s1 = Integer.toString(123); // 정수 123을 문자열 "123" 으로 변환  
String s2 = Integer.toHexString(123); // 정수 123을 16진수의 문자열 "7b"로 변환  
String s3 = Float.toString(3.141592f); // 실수 3.141592를 문자열 "3.141592"로 변환  
String s4 = Character.toString('a'); // 문자 'a'를 문자열 "a"로 변환  
String s5 = Boolean.toString(true); // 불린 값 true를 문자열 "true"로 변환
```

박싱과 언박싱

- 박싱(boxing)
 - 기본 타입의 값을 Wrapper 객체로 변환하는 것
- 언박싱(unboxing)
 - Wrapper 객체에 들어 있는 기본 타입의 값을 빼내는 것



예전에는 그림처럼
박싱/언박싱 할 때
생성자, 메소드 호출했음

자동 박싱/자동 언박싱

- JDK 1.5부터 지원
- 자동 박싱(Auto boxing): 기본 타입의 값을 자동으로 Wrapper 객체로 변환
- 자동 언박싱(Auto unboxing): Wrapper 객체를 자동으로 기본 타입 값으로 변환

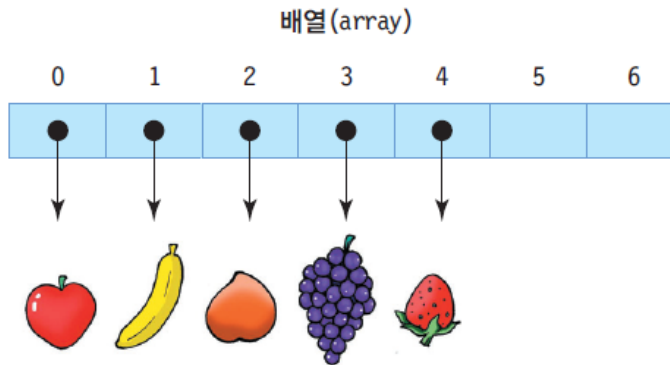
```
<%  
int i = 10;  
Integer ten = i; // auto boxing  
out.print("ten = " + ten + "<br>");  
  
i = ten + 10; // auto unboxing  
out.print("i = " + i);  
%>
```

```
ten = 10  
i = 20
```


다음으로, 제네릭/컬렉션에 대해,
그리고 ArrayList에 대해
간단히 소개하겠습니다!!

컬렉션(collection)

- 컬렉션
 - 요소(element)라고 불리는 가변 개수의 객체들의 저장소
 - 객체들의 컨테이너라고도 불림
 - 요소의 개수에 따라 크기 자동 조절
 - 요소의 삽입, 삭제에 따른 요소의 위치 자동 이동
 - 고정 크기의 배열을 다루는 어려움 해소
 - 다양한 객체들의 삽입, 삭제, 검색 등의 관리 용이



- 고정 크기 이상의 객체를 관리할 수 없다.
- 배열의 중간에 객체가 삭제되면 응용프로그램에서 자리를 옮겨야 한다.

컬렉션(collection)



- 가변 크기로서 객체의 개수를 염려할 필요 없다.
- 컬렉션 내의 한 객체가 삭제되면 컬렉션이 자동으로 자리를 옮겨준다.

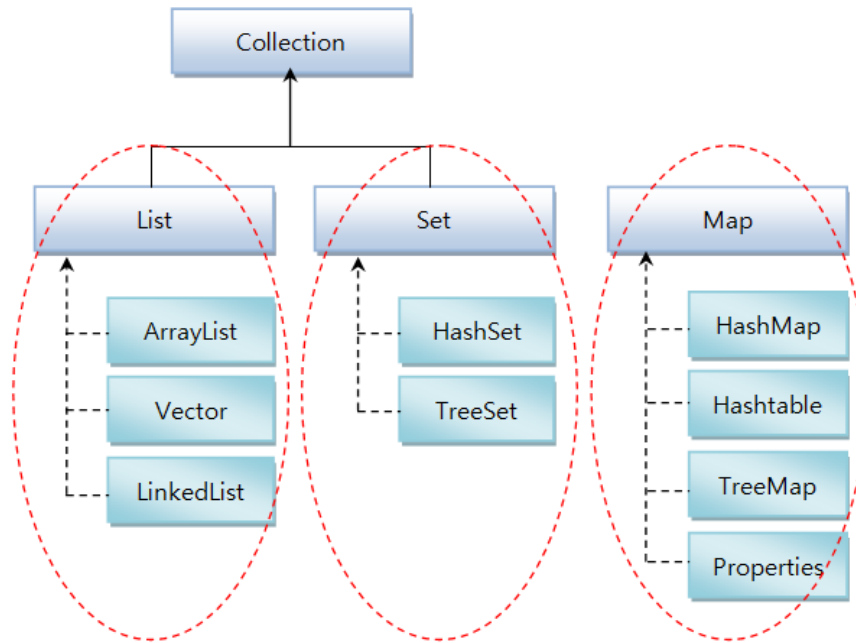
컬렉션 프레임워크(Collection Framework)

- 배열의 문제점: 저장할 수 있는 객체 수가 배열을 생성할 때 결정
 - 불특정 다수의 객체를 저장하기에는 문제. 너무 넉넉하게 잡아 두자니 메모리 낭비, 그렇다고 적게 잡았다가 나중에 모자라면 낭패
 - 그리고, 객체 삭제했을 때 해당 인덱스가 비게 됨
 - 낱알 빠진 옥수수 같은 배열
 - 객체를 저장하려면 어디가 비어있는지 확인해야
- 컬렉션 라이브러리는 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 제공
 - java.util 패키지에 포함

배열

0	1	2	3	4	5	6	7	8	9
●	●	×	●	×	●	×	●	●	×

컬렉션 프레임워크의 주요 인터페이스



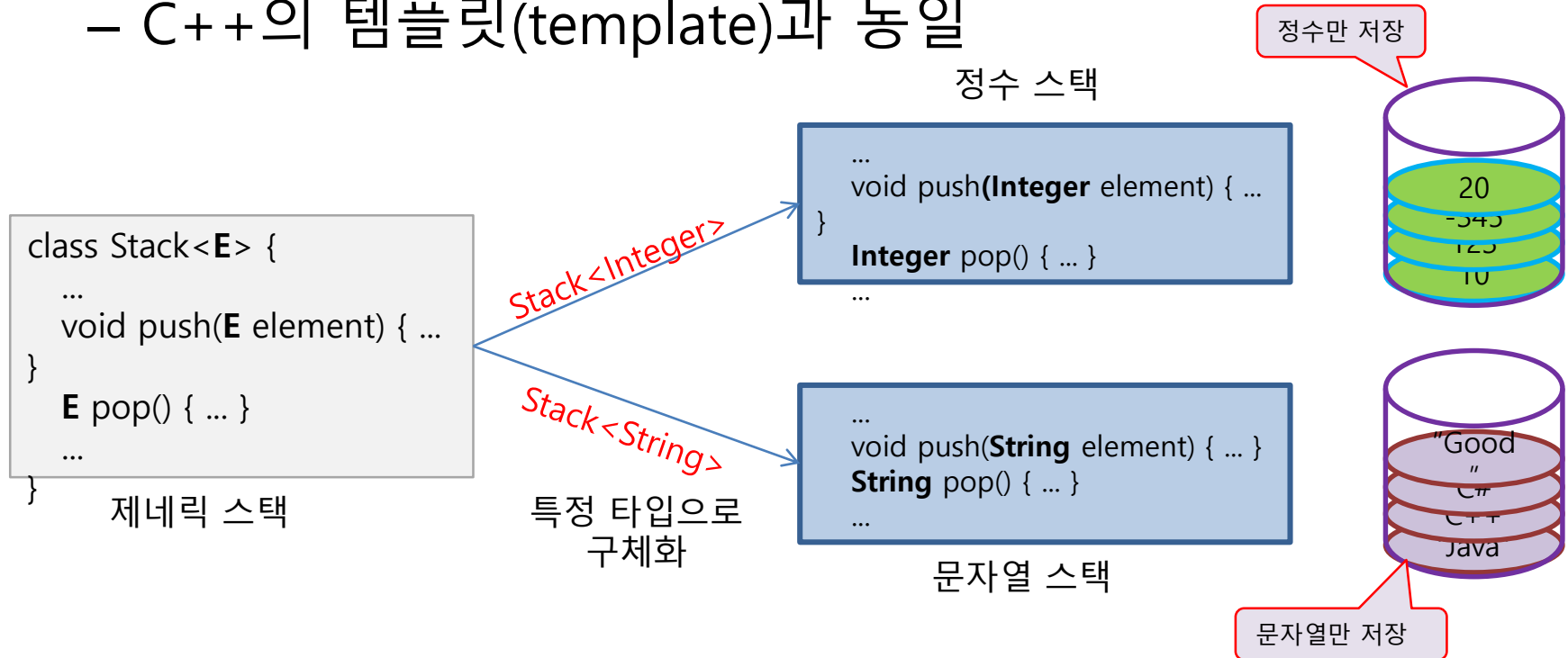
인터페이스 분류		특징	구현 클래스
Collection	List 계열	<ul style="list-style-type: none"> - 순서를 유지하고 저장 - 중복 저장 가능 	ArrayList, Vector, LinkedList
	Set 계열	<ul style="list-style-type: none"> - 순서를 유지하지 않고 저장 - 중복 저장 안됨 	HashSet, TreeSet
Map 계열		<ul style="list-style-type: none"> - 키와 값의 쌍으로 저장 - 키는 중복 저장 안됨 	HashMap, Hashtable, TreeMap, Properties

컬렉션과 제네릭

- 컬렉션은 제네릭(generics) 기법으로 구현됨
- 컬렉션의 요소는 객체만 가능했었는데, JDK 1.5부터 자동 박싱/언박싱 기능으로 기본 타입 사용 가능
- 제네릭
 - ▣ 특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화시키는 기법
 - `<T>`, `<E>`, `<K>`, `<V>` : 타입 매개 변수
 - 요소 타입을 일반화한 타입
 - ▣ 제네릭 클래스 사례
 - 제네릭 벡터 : `Vector<E>`
 - E에 특정 타입으로 구체화
 - 정수만 다루는 벡터 `Vector<Integer>`
 - 문자열만 다루는 벡터 `Vector<String>`

제네릭의 기본 개념

- 모든 종류의 데이터 타입을 다룰 수 있도록 일반화된 타입 매개 변수로 클래스나 메소드를 작성하는 기법
 - C++의 템플릿(template)과 동일



List 컬렉션

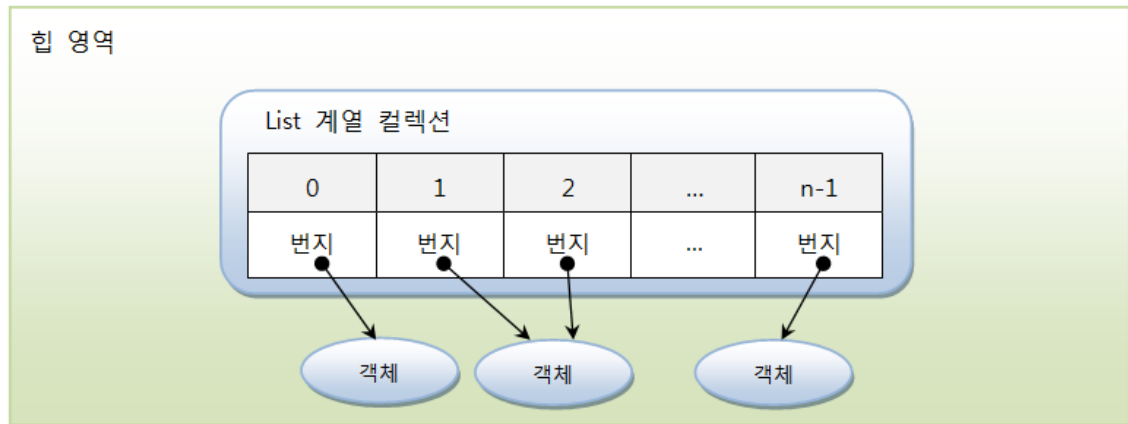
❖ List 컬렉션의 특징 및 주요 메소드

■ 특징

- 인덱스로 관리
- 중복해서 객체 저장 가능

■ 구현 클래스

- ArrayList
- Vector
- LinkedList



List 컬렉션

❖ List 컬렉션의 특징 및 주요 메소드

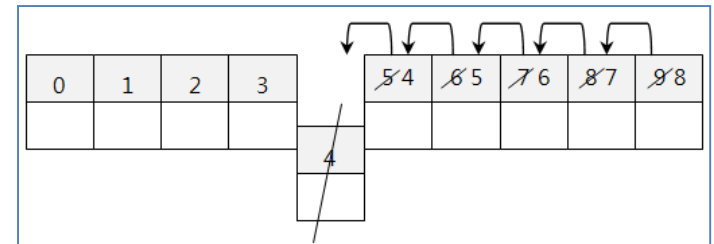
■ 주요 메소드

기능	메소드	설명
객체 추가	<code>boolean add(E e)</code>	주어진 객체를 맨끝에 추가
	<code>void add(int index, E element)</code>	주어진 인덱스에 객체를 추가
	<code>set(int index, E element)</code>	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 검색	<code>boolean contains(Object o)</code>	주어진 객체가 저장되어 있는지 여부
	<code>E get(int index)</code>	주어진 인덱스에 저장된 객체를 리턴
	<code>isEmpty()</code>	컬렉션이 비어 있는지 조사
	<code>int size()</code>	저장되어있는 전체 객체수를 리턴
객체 삭제	<code>void clear()</code>	저장된 모든 객체를 삭제
	<code>E remove(int index)</code>	주어진 인덱스에 저장된 객체를 삭제
	<code>boolean remove(Object o)</code>	주어진 객체를 삭제

List 컬렉션

❖ ArrayList

- 저장 용량
 - 초기 용량 : 10 (따로 지정 가능)
 - 저장 용량을 초과한 객체들이 들어오면
 - 자동적으로 늘어남. 고정도 가능
- 객체 제거
 - 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨짐



ArrayList<E>

□ ArrayList<E>의 특성

- ▣ java.util.ArrayList, 가변 크기 배열을 구현한 클래스
 - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
- ▣ ArrayList에 삽입 가능한 것
 - 객체, null
 - 기본 타입은 박싱/언박싱으로 Wrapper 객체로 만들어 저장
- ▣ ArrayList에 객체 삽입/삭제
 - 리스트의 맨 뒤에 객체 추가
 - 리스트의 중간에 객체 삽입
 - 임의의 위치에 있는 객체 삭제 가능
- ▣ 벡터와 달리 스레드 동기화 기능 없음
 - 다수 스레드가 동시에 ArrayList에 접근할 때 동기화되지 않음
 - 개발자가 스레드 동기화 코드 작성

ArrayList<E>

□ 중간고사 전에 쇼핑몰 ver. 2 기억하시나요?

(기억 안 나도 괜찮습니다. 곧 다시 만납니다~)

▣ 고객이 산 물건을 배열에 저장하는 기능

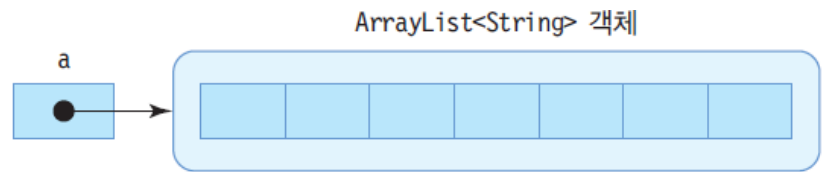
- 고객이 얼마나 통 크게 쇼핑하실라나? 알 수가 없음
- radio도 아니고 checkbox로 마구마구 장바구니 넣으시는 고객분들~
 - 환영합니다!!
- 모든 고객마다 배열 완전 크게 잡아 놓으면, 메모리 낭비
- 대충 적당히 잡아 봤다가는 통 큰 고객 놓침 -_-;;
 - 여기부터 저기까지 다 주세요~ 하는 고객
- 게다가 긴 장바구니에서 목록에서 불필요한 거 자꾸 삭제하면, 일반 배열이라면?

ArrayList<E> 클래스의 주요 메소드

메소드	설명
<code>boolean add(E element)</code>	ArrayList의 맨 뒤에 <code>element</code> 추가
<code>void add(int index, E element)</code>	인덱스 <code>index</code> 에 지정된 <code>element</code> 삽입
<code>boolean addAll(Collection<? extends E> c)</code>	컬렉션 <code>c</code> 의 모든 요소를 ArrayList의 맨 뒤에 추가
<code>void clear()</code>	ArrayList의 모든 요소 삭제
<code>boolean contains(Object o)</code>	ArrayList가 지정된 객체를 포함하고 있으면 <code>true</code> 리턴
<code>E elementAt(int index)</code>	<code>index</code> 인덱스의 요소 리턴
<code>E get(int index)</code>	<code>index</code> 인덱스의 요소 리턴
<code>int indexOf(Object o)</code>	<code>o</code> 와 같은 첫 번째 요소의 인덱스 리턴. 없으면 <code>-1</code> 리턴
<code>boolean isEmpty()</code>	ArrayList가 비어 있으면 <code>true</code> 리턴
<code>E remove(int index)</code>	<code>index</code> 인덱스의 요소 삭제
<code>boolean remove(Object o)</code>	<code>o</code> 와 같은 첫 번째 요소를 ArrayList에서 삭제
<code>int size()</code>	ArrayList가 포함하는 요소의 개수 리턴
<code>Object[] toArray()</code>	ArrayList의 모든 요소를 포함하는 배열 리턴

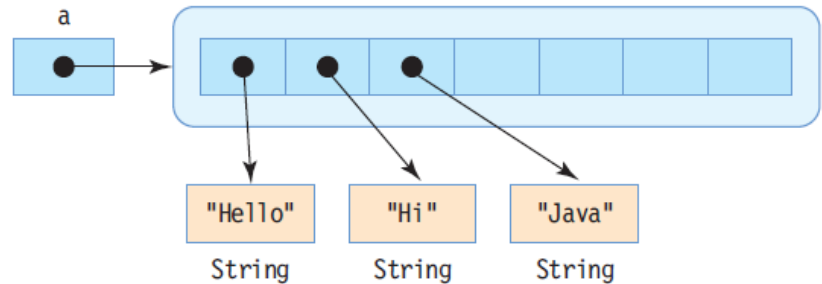
ArrayList 생성

```
ArrayList<String> a = new ArrayList<String>(7);
```



요소 삽입

```
a.add("Hello");  
a.add("Hi");  
a.add("Java");
```



요소 개수 n
용량

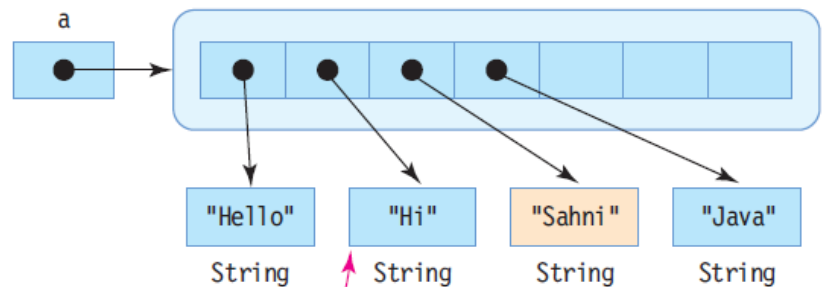
```
int n = a.size(); // n은 3  
int c = a.capacity(); // capacity() 메소드 없음
```

n = 3

요소 중간 삽입

```
a.add(2, "Sahni");
```

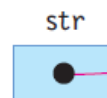
```
a.add(5, "Sahni");  
// a.size()보다 큰 위치에 삽입 불가능, 오류
```



요소 알아내기

```
String str = a.get(1);
```

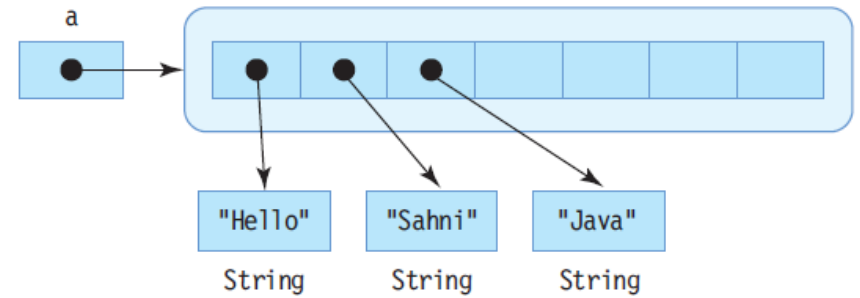
"Hi"



요소 삭제

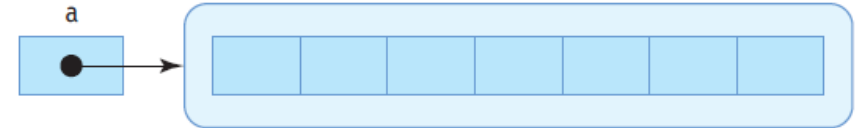
```
a.remove(1);
```

```
a.remove(4); // 오류
```



모든 요소 삭제

```
a.clear();
```



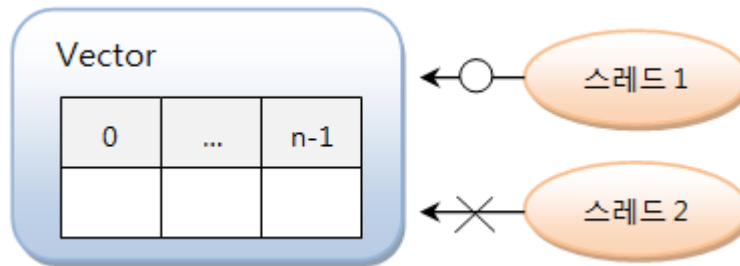
List 컬렉션

❖ Vector

```
List<E> list = new Vector<E>();
```

■ 특징

- Vector는 스레드 동기화(synchronization)
 - 복수의 스레드가 동시에 Vector에 접근해 객체를 추가, 삭제 하더라도 스레드에 안전(thread safe)



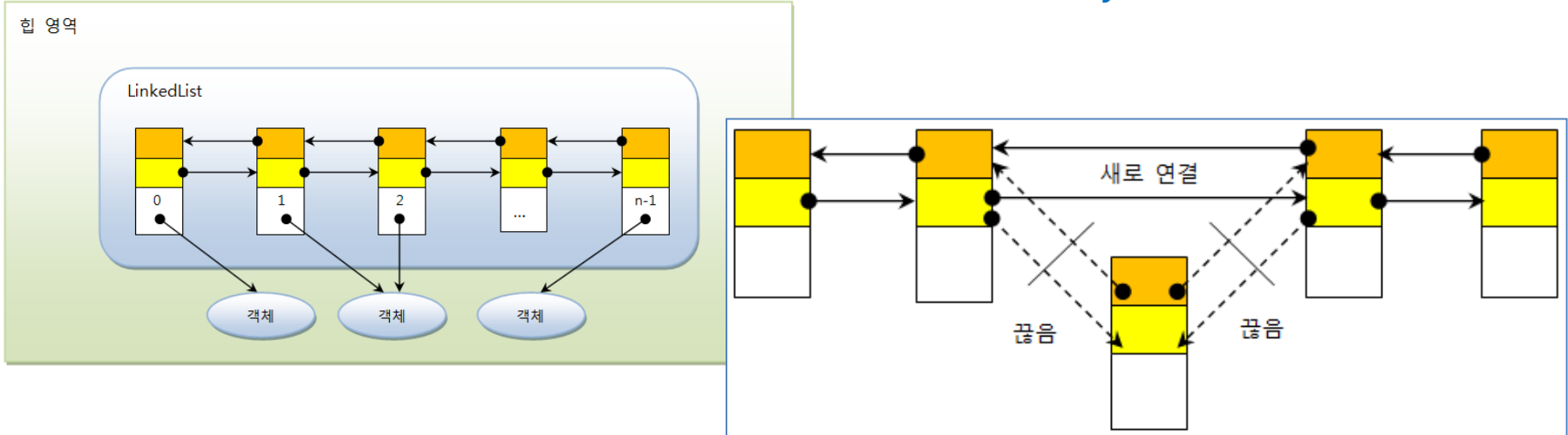
List 컬렉션

❖ LinkedList

```
List<E> list = new LinkedList<E>();
```


■ 특징

- 인접 참조를 링크해서 체인처럼 관리
- 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤 링크만 변경
- 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 좋은 성능



ArrayList를 이용한 쇼핑몰 장바구니 기능

menu.jsp



```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.ArrayList" %>
<%
if (session.getAttribute("car_s") == null) {
    ArrayList<String> arr = new ArrayList<String>(3);
    session.setAttribute("car_s", arr);
}
%>
<body style="background-color:lime">
<h2>메뉴</h2>
<ul>
    <li><a href = "car.jsp">람보르기니와 프라임</a></li>
    <li><a href = "jang.jsp">장바구니</a></li>
    <li><a href = "logout.jsp">logout</a></li>
</ul>
</body>
```

메뉴

- 람보르기니와 프라임
- 장바구니
- logout

```
ArrayList<String> arr = new ArrayList<String>(3);
session.setAttribute("car_s", arr);
```

편의상 menu.jsp에 넣었습니다.
원래는 이 두 줄을 check_id.jsp에서
세션에 id넣을 때 함께 넣어주면 됩니다~

어떤 차를 원하세요? (멀티 선택 가능)



☐ 노랑이(무르시엘라고)



☐ 파랑이(아벤타도르)



☐ 레드(베네노)



☐ 옵티머스 프라임도 차 맞음

장바구니에 저장

[메뉴로 돌아가기](#)

car.jsp

[illegible]

save_car.jsp : 선택한 차를 세션에 넣음

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.ArrayList" %>
<%
request.setCharacterEncoding("UTF-8");
String[] car = request.getParameterValues("car_n"); // 앞에서 선택한 차들
if(car != null) {
    ArrayList<String> arr = (ArrayList<String>)session.getAttribute("car_s");
    for(String s:car) {
        arr.add(s);
    }
    session.setAttribute("car_s", arr);
}
response.sendRedirect("jang.jsp");
%>
```

- (1) 세션에 있는 *ArrayList* 가져와서
- (2) 그 *ArrayList*에 새로 선택한 거 *add*한 후
- (3) 다시 세션에 넣음

1번과정 생략하면 어떻게 될까요?

jang.jsp : 세션에 있는 거 보여줌

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.ArrayList" %>
<body style="background-color:yellow">
<h4>장바구니 품목 리스트</h4>
<%
ArrayList<String> arr = (ArrayList<String>)session.getAttribute("car_s");
%>
<form method="post" action="delete.jsp">
<%for (int i=0; i<arr.size(); i++) {%>
    <input type="radio" name="delno_n" value="%i%" /> <%=arr.get(i)%> <br>
<%}%>
<p>
<input type="submit" value="선택 항목 삭제"> &nbsp;&nbsp;&nbsp;
<a href="jang.jsp">전체 구매 (나중 추가)</a> <p>
<a href = "menu.jsp"> 메뉴로 돌아가기 </a>
</form>
</body>
```

장바구니 품목 리스트

- 파랑이(아벤타도르)
- 레드(베네노)

선택항목삭제

전체 구매 (나중 추가)

[메뉴로 돌아가기](#)

"save_car.jsp 를 따로 두지 않고
그 내용을 jang.jsp 안에 넣으면
어떤 문제가 생길까요?"

delete.jsp : 한 항목 삭제

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.ArrayList" %>
<%
int i = Integer.parseInt(request.getParameter("delno_n"));
ArrayList<String> arr = (ArrayList<String>)session.getAttribute("car_s");

arr.remove(i);
session.setAttribute("car_s",arr);

response.sendRedirect("jang.jsp");
%>
```

장바구니 품목 리스트

☐ 레드(베네노)

선택항목삭제

[전체 구매 \(나중 추가\)](#)

[메뉴로 돌아가기](#)

logout.jsp : 로그아웃

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<% session.invalidate(); %>
<script>
alert("logout되었습니다. 메뉴로 돌아갑니다.");
location.href="menu.jsp";
</script>
```